SAYABIDEVS AI AND DATASCIENCE

TEAM # 4

AI POWERED IMAGE RECOGNITION SYSTEM

SUBMITTED BY:

Anas Ansari

Abdullah Hashim

Sabih Ur Rehman

Ifra Siddiqui

TASK DESCRIPTION:

- Build and fine-tune a deep learning model for image recognition.
- Utilize transfer learning by starting with a pre-trained model and adapting it to your specific task.
- Learn how to preprocess and augment image data to improve model performance.
- Gain insights into evaluating model accuracy and making improvements.
- Gather a dataset of images for your classification task. You can use publicly available datasets or create your own.
- Preprocess the images by resizing, normalizing, and augmenting the data to increase model robustness.
- Choose a pre-trained deep learning model from popular frameworks like TensorFlow or PyTorch (e.g., ResNet, MobileNet, VGG).
- Adapt the pre-trained model to your specific classification task by modifying the output layer for the desired number of classes.
- Split your dataset into training, validation, and test sets.
- Train the model using the training set and validate it using the validation set.
- Implement model evaluation metrics (e.g., accuracy, precision, recall) to assess model performance.
- Experiment with hyperparameters, such as learning rate, batch size, and optimizer, to improve model convergence.
- Fine-tune the model based on validation performance.
- If you're comfortable, deploy the trained model as a simple web application or API using frameworks like Flask or FastAPI.
- TensorFlow Tutorials: TensorFlow Tutorials
- PyTorch Tutorials: PyTorch Tutorials
- Transfer Learning with a Pre-trained ConvNet: PyTorch Transfer Learning Tutorial
- TensorFlow Image Preprocessing: Image Preprocessing in TensorFlow
- PyTorch Data Augmentation: Data Augmentation in PyTorch
- Understanding Classification Metrics: Classification Metrics Explained

INTRODUCTION:

In the world of artificial intelligence, image recognition has gained significant importance due to its applications in various fields, including healthcare, security, and entertainment. This report presents a project that utilizes AI-powered image recognition to classify images of cats and dogs. The project leverages transfer learning, utilizing a pre-trained MobileNetV2 model to achieve accurate classification results. This report delves into the approach, model architecture, training process, evaluation results, challenges faced, and concludes with an overview of the project's achievements.

APPROACH:

Our project adopts a strategic approach that encompasses a series of pivotal steps, each playing an instrumental role in achieving the overarching objective:

DATASET LOADING: The project capitalizes on the widely used Cats and Dogs Dataset, housing two distinct classes - cats and dogs. The dataset is meticulously loaded, and all images are standardized to a uniform resolution of 100x100 pixels.

DATA AUGMENTATION AND PREPROCESSING: In a bid to enhance the model's ability to generalize, data augmentation techniques are judiciously applied. Ranging from rotations to width and height shifts, and horizontal flipping, these techniques augment the training dataset, fostering improved generalization.

Performing Data augmentation and preprocessing

Performing augmentation using data generator.

In [16]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [17]:
```
datagen = ImageDataGenerator(rescale = 1.0/255, rotation_range = 20, width_shift_range = 0.2, height_shift_range = 0.2,
                             horizontal_flip = True)
```

In [22]:
```
images = []
labels = []

# Iterate through class directories
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    if os.path.isdir(class_dir):
        class_label = class_name  # Use class name as label
        for img_filename in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_filename)
            try:
                img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
                img_array = tf.keras.preprocessing.image.img_to_array(img)
                images.append(img_array)
                labels.append(class_name)  # Use class name as label
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")

print(f"Number of successfully loaded images: {len(images)}")
```

```
Error loading image D:\cats_vs_dogs\cat\666.jpg: cannot identify image file <_io.BytesIO object at 0x0000018C141931F0>
Error loading image D:\cats_vs_dogs\cat\Thumbs.db: cannot identify image file <_io.BytesIO object at 0x0000018C140777E0>
Error loading image D:\cats_vs_dogs\dog\11702.jpg: cannot identify image file <_io.BytesIO object at 0x0000018C140777E0>
C:\Users\anasj\anaconda3\lib\site-packages\PIL\TiffImagePlugin.py:858: UserWarning: Truncated File Read
  warnings.warn(str(msg))
Error loading image D:\cats_vs_dogs\dog\Thumbs.db: cannot identify image file <_io.BytesIO object at 0x0000018C14046160>
Number of successfully loaded images: 24998
```

STRATEGIC DATASET SPLITTING: The dataset undergoes a meticulous division into three critical subsets - training, validation, and test sets. This segmentation sets the stage for subsequent training and evaluation steps. The class labels undergo transformation through label encoding and are further subjected to one-hot encoding to facilitate model comprehension.

HARNESSING A PRE-TRAINED MODEL: The art of transfer learning comes to fruition as the pre-trained MobileNetV2 model is integrated. The pre-existing architecture undergoes augmentation through the infusion of global average pooling and fully connected layers, tailored to suit the unique classification task.

Loading a Pre-trained model

Loading MobileNetV2 model By using the initial layers of the pre-trained model we are implementing "TRANSFER LEARNING" as we are rejecting the top layers which include the fully connected layers of MobileNetV2 trained on ImageNet dataset. In simpler terms we are taking only the architecture and the weights of the said model.

In [10]:
```
# Load a pre-trained MobileNetV2 model
base_model = MobileNetV2(input_shape=(100, 100, 3), include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_of_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input sh
ape (224, 224) will be loaded as the default.
```

DILIGENT MODEL TRAINING: The core of the project lies in the thorough training regimen. The model is trained using the training dataset and is subjected to rigorous validation using the validation dataset. Training involves refining the model's weights through the strategic application of the Adam optimizer while minimizing the categorical cross-entropy loss. The model's efficacy is quantified through the prism of accuracy.

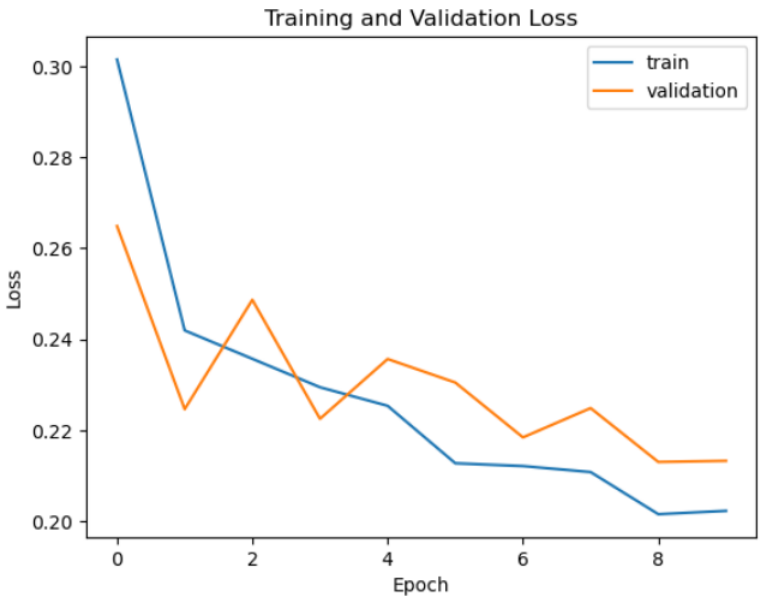
IN-DEPTH EVALUATION AND REPORTING: The true measure of the model's mettle is assessed through evaluation on the test dataset. The results are not merely restricted to a binary distinction. The precision, recall, and F1-score are meticulously computed, offering a nuanced glimpse into the model's performance. A comprehensive classification report is meticulously generated, encapsulating the model's prowess.

MODEL ARCHITECTURE AND TRAINING

The project employs transfer learning with a MobileNetV2 model. The top layers of the pre-trained MobileNetV2 are removed, and two additional dense layers are added. The model is compiled with the Adam optimizer and categorical cross-entropy loss. During training, data augmentation techniques are applied to improve the model's robustness. The training process is performed over ten epochs, and training and validation loss curves are plotted to visualize model performance.

Plotting training and validation loss

In [12]:
```
# Plot Training and Validation Loss Curves
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



EVALUATION RESULTS

The model's evaluation results indicate its effectiveness in classifying images of cats and dogs. The classification report demonstrates high precision, recall, and F1-score for both classes. The model achieves an impressive accuracy of 92% on the test dataset, highlighting its capability to distinguish between cats and dogs accurately.

Generating Classification report

In [15]:
```
from sklearn.metrics import classification_report

# Print classification report
class_labels = list(test_generator.class_indices.keys())
report = classification_report(y_true, y_pred_classes, target_names=class_labels)
print(report)
```

```
              precision    recall  f1-score   support

         cat       0.91      0.93      0.92     12499
         dog       0.93      0.90      0.92     12499

    accuracy                           0.92     24998
   macro avg       0.92      0.92      0.92     24998
weighted avg       0.92      0.92      0.92     24998
```

Prediction 1

In [4]:
```
import cv2
import numpy as np

img_size = (100, 100)  # Same as the image size used during training

# Load and preprocess the external image
external_image_path = 'D:\cattest.jpg'
img = cv2.imread(external_image_path)
img_resized = cv2.resize(img, img_size)
external_input = np.expand_dims(img_resized / 255.0, axis=0)  # Preprocess and normalize the image

# Load the external image
external_img1 = mpimg.imread(external_image_path)

# Display the image
plt.figure(figsize=(6, 6))
plt.imshow(external_img1)
plt.title('prediction 1')
plt.axis('off')
plt.show()
```

In [5]:
```
predictions = model.predict(external_input)


predicted_class_index = np.argmax(predictions[0])
class_labels = ['Cat', 'Dog']
predicted_class_label = class_labels[predicted_class_index]

print("Predicted Class:", predicted_class_label)
```

```
1/1 [==============================] - 0s 410ms/step
Predicted Class: Cat
```

CHALLENGES FACED

During the project's development, several challenges were encountered:

DATA QUALITY: Improper and noisy data in the dataset was causing the prevention of loading the image dataset, the improper images and thumbnail files were identified and were removed from the dataset manually.

HYPERPARAMETER TUNING: The Model was initially trained on 30 epochs, since the graph indicates, that after 15 epochs, model started overfitting, the number of epochs were reduced to 15, then to 10 for better generalization.

MODEL COMPLEXITY: OOM (Out of Memory) error was encountered as initially 32 batches of 25000, image of size 224 by 224 was taken. To handle this issue Image size was reduced to 100 by 100, and batch sized were reduced to 8.

CONCLUSION:

In conclusion, this project demonstrates the successful implementation of an AI-powered image recognition system using transfer learning and data augmentation. The model effectively classifies images of cats and dogs with an accuracy of 92%, showcasing the potential of deep learning in solving real-world image classification tasks. The use of transfer learning significantly accelerates model development and enhances classification accuracy. The project's outcomes contribute to the field of computer vision and offer practical insights into building robust image recognition systems.